Spring 2005 Option III Course

EE382C.3 Verification and Validation

Instructor

Sarfraz Khurshid
ACES 5.120
(512)-471-8244
khurshid@ece.utexas.edu

Time and Location

All lectures will be 1:00-5:00pm. The location will be as follows.

Jan 21-22	TCC 2.110
Feb 18-19	TCC 2.110
Mar 11-12	TCC 1.124
Apr 15-16	TCC 2.110
May 13-14	TCC 2.110

Prerequisites

The students are expected to have basic knowledge of data structures and object-oriented programming, and considerable programming experience.

Outline

The process of software validation includes reasoning about (the correctness of) programs, whether formally—a process that is termed verification—or informally, and testing programs. This course focuses on verification and testing. The course is organized as a series of research/tool paper presentations and discussions. The selected papers will cover traditional and state-of-the-art techniques for software validation. (See the References Section for a list of candidate papers. Different papers may be selected in view of class preferences.) The course content will cover both techniques for dynamic analysis, such as glass box and black box testing, equivalence partitioning, boundary value analysis,

test strategy and automation, regression testing and debugging, and techniques for static analysis, such as shape analysis, and also techniques for software model checking including those that employ artificial intelligence based heuristics.

Grading

The grade will be based on class participation (10%), homeworks and quizes (50%) and a final group project (40%). Students must participate actively in the class. The final project will be done in a group of three or four students. A typical project would involve performing a case study using a tool studied in the class. With instructor's permission, the students may choose to work on a suitable idea of their own. Good projects will result in work that is of a quality expected for conference/workshop publication. At the end of the course, students will present their projects to the class.

Background Reading

The following texts provide some basic material that would help students understand the more advanced material in papers:

1. Model Checking by Edmund M. Clarke, Orna Grumberg and Doron A. Peled. ISBN: 0262032708
2. A Practitioner's Guide to Software Test Design by Lee Copeland. ISBN: 158053791X

References

[1] Mike Barnett, Robert DeLine, Manuel Fahndrich, K. Rustan M. Leino, and Wolfram Schulte. Verification of object-oriented programs with invariants. In *Proc. ECOOP 2003 Workshop on Formal Techniques for Java-like Programs (FTfJP)*, 2003.

[2] Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: An overview. In *Proc. Cassis International Workshop*, 2004. to appear.

[3] Mike Barnett and Wolfram Schulte. Runtime verification of .NET contracts. *Journal of Systems and Software*, 65(3), 2003.

[4] Dirk Beyer, Adam J. Chlipala, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. Generating tests from counterexamples. In *Proc. 26th International Conference on Software Engineering (ICSE)*, pages 326–335, 2004.

[5] Chandrasekhar Boyapati, Sarfraz Khurshid, and Darko Marinov. Korat: Automated testing based on Java predicates. In *Proc. International Symposium on Software Testing and Analysis (ISSTA)*, pages 123–133, July 2002.

[6] William R. Bush, Jonathan D. Pincus, and David J. Sielaff. A static analyzer for finding dynamic programming errors. *Software—Practice and Experience*, 30(7):775–802, 2000.

[7] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, 1997.

[8] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole. Buffer overflows: Attacks and defenses for the vulnerability of the decade. In *Proc. DARPA Information Survivability Conference and Exposition (DISCEX)*, January 2000.

[9] Brian Demsky and Martin Rinard. Automatic detection and repair of errors in data structures. In *Proc. ACM SIGPLAN 2003 Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 78–95, 2003.

[10] Cormac Flanagan, K. Rustan M. Leino, Mark Lillibridge, Greg Nelson, James B. Saxe, and Raymie Stata. Extended static checking for Java. In *Proc. ACM SIGPLAN 2002 Conference on Programming language design and implementation*, pages 234–245, 2002.

[11] Stefan M. Freudenberger, Jacob T. Schwartz, and Micha Sharir. Experience with the SETL optimizer. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(1), 1983.

[12] Patrice Godefroid. Model checking for programming languages using VeriSoft. In *Proc. 24th Annual ACM Symposium on the Principles of Programming Languages (POPL)*, pages 174–186, Paris, France, January 1997.

[13] Patrice Godefroid and Sarfraz Khurshid. Exploring very large state spaces using genetic algorithms. In *Proc. 8th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Grenoble, France, April 2002.

[14] Sudheendra Hangal and Monica S. Lam. Tracking down software bugs using automatic anomaly detection. In *Proc. 24th International Conference on Software Engineering (ICSE)*, May 2002.

[15] Johannes Henkel and Amer Diwan. Discovering algebraic specifications from java classes. In *Proc. European Conference on Object-Oriented Programming (ECOOP)*, July 2003.

[16] Johannes Henkel and Amer Diwan. A tool for writing and debugging algebraic specifications. In *Proc. 26th International Conference on Software Engineering (ICSE)*, 2004.

[17] Daniel Jackson. Micromodels of software: Modelling and analysis with Alloy, 2001. `http://sdg.lcs.mit.edu/alloy/book.pdf`.

[18] Sarfraz Khurshid and Darko Marinov. TestEra: Specification-based testing of Java programs using SAT. *Automated Software Engineering Journal*, 2004.

[19] Sarfraz Khurshid, Corina Pasareanu, and Willem Visser. Generalized symbolic execution for model checking and testing. In *Proc. 9th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Warsaw, Poland, April 2003.

[20] Todd Millstein. Practical predicate dispatch. In *Proc. ACM SIGPLAN 2004 Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, October 2004.

[21] J. Strother Moore and George Porter. The apprentice challenge. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(3), 2002.

[22] M. Musuvathi and D. Engler. Model checking large network protocol implementations. In *Proc. First Symposium on Networked Systems Design and Implementation*, pages 155–168, 2004.

[23] Robby, Edwin Rodríguez, Matthew Dwyer, and John Hatcliff. Checking strong specifications using an extensible software model checking framework. In *Proc. 10th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, 2004.

[24] Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, and Thomas Anderson. Eraser: A dynamic data race detector for multithreaded programs. *ACM Transactions on Computer Systems*, 15(4):391–411, 1997.

[25] Emin Gün Sirer and Brian N. Bershad. Using production grammars in software testing. In *Proc. 2nd conference on Domain-specific languages*, pages 1–13, 1999.

[26] Willem Visser, Klaus Havelund, Guillaume Brat, and SeungJoon Park. Model checking programs. In *Proc. 15th IEEE International Conference on Automated Software Engineering (ASE)*, Grenoble, France, 2000.

[27] Christoph von Praun and Thomas R. Gross. Static conflict analysis for multithreaded object-oriented programs. In *Proc. ACM SIGPLAN'03 Conference on Programming Language Design and Implementation (PLDI)*, June 2003.

[28] John Whaley, Michael C. Martin, and Monica S. Lam. Automatic extraction of object-oriented component interfaces. In *Proc. International Symposium on Software Testing and Analysis (ISSTA)*, July 2002.

[29] Andreas Zeller. Yesterday, my program worked. today, it does not. why? In *Proc. 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 253–267, 1999.

[30] Andreas Zeller. Isolating cause-effect chains from computer programs. In *Proc. 10th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 1–10, 2002.